A nested heuristic for parameter tuning in Support Vector Machines

Emilio Carrizosa
Universidad de Sevilla (Spain)
ecarrizosa@us.es
Belén Martín-Barragán
The University of Edinburgh (United Kingdom)
Belen.Martin@ed.ac.uk
Dolores Romero Morales
University of Oxford (United Kingdom)
dolores.romero-morales@sbs.ox.ac.uk

October 15, 2013

Abstract

The default approach for tuning the parameters of a Support Vector Machine (SVM) is a grid search in the parameter space. Different metaheuristics have been recently proposed as a more efficient alternative, but they have only shown to be useful in models with a low number of parameters. Complex models, involving many parameters, can be seen as extensions of simpler and easy-to-tune models, yielding a nested sequence of models of increasing complexity. In this paper we propose an algorithm which successfully exploits this *nested* property, with two main advantages versus the state of the art. First, our framework is general enough to allow one to address, with the very same method, several popular SVM parameter models encountered in the literature. Second, as algorithmic requirements we only need either an SVM library or any routine for the minimization of convex quadratic functions under linear constraints. In the computational study, we address Multiple Kernel Learning tuning problems for which grid search clearly would be infeasible, while our classification accuracy is comparable to that of ad-hoc model-dependent benchmark tuning methods.

Keywords: supervised classification, Support Vector Machines, parameter tuning, nested heuristic, Variable Neighborhood Search, Multiple Kernel Learning.

1 Introduction

Support Vector Machines (SVM) [4, 9, 15, 46, 47] is a Supervised Classification technique rooted in Statistical Learning Theory [46, 47], whose success is based on the ability of building nonlinear classifiers.

Let Ω denote a data set of n records, each associated with a pair (x^i, y^i), with $x^i \in I\!R^d$ (the predictor vector of record i) and $y^i \in \{-1, 1\}$ (the label of record i). The SVM classifier will classify records with predictor vectors $x \in I\!R^d$ by means of a score $s(x)$ of the form

$$s(x) = \sum_{i=1}^{n} \alpha^i y^i K(x, x^i), \tag{1}$$

where $K : I\!R^d \times I\!R^d \to I\!R$ is the so-called SVM *kernel*, see [15, 26, 27] and references therein, and the coefficients α^i are obtained by solving the following concave quadratic maximization problem with box constraints plus one linear constraint:

$$\begin{array}{ll} \max & \sum_{i=1}^{n} \alpha^i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha^i \alpha^j y^i y^j K(x^i, x^j) \\ \text{s.t.} & \sum_{i=1}^{n} \alpha^i y^i = 0 \\ & \alpha \in [0, C]^n. \end{array} \tag{2}$$

Here $C > 0$ is the so-called regularization parameter which bounds the influence of each record i in the score function s. It is well-known that the choice of both the kernel K and the regularization parameter C is crucial to the SVM classification accuracy, [32]. For this reason, *tuning* (i.e., choosing) the SVM parameters becomes a fundamental yet nontrivial issue. Designing *simple* and *effective* tuning procedures will be useful for the wide variety of practitioners using SVM.

In order to formulate the SVM parameter tuning problem, note that, setting $\vartheta^i = \frac{\alpha^i}{C}$ in (2), we obtain the equivalent problem

$$\begin{array}{ll} \max & \sum_{i=1}^{n} \vartheta^i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \vartheta^i \vartheta^j y^i y^j C K(x^i, x^j) \\ \text{s.t.} & \sum_{i=1}^{n} \vartheta^i y^i = 0 \\ & \vartheta \in [0, 1]^n. \end{array} \tag{3}$$

From this formulation it is clear that the classifier obtained using either (2) or (3) depends on C and K through its product CK. Tuning $C > 0$ and K in a given class of kernels $\mathcal{K}_0$ is therefore equivalent to selecting K in the conic hull of $\mathcal{K}_0$, $\mathcal{K} = \{CK : C > 0, K \in \mathcal{K}_0\}$.

Ideally, K should be chosen by maximizing $a(K)$, the probability of correct classification of incoming records if one classifies following the classifier obtained from (1). Since the SVM theory makes no distributional assumptions on the incoming data, $a(\cdot)$ cannot be evaluated, and, instead, an estimate $\hat{a}(\cdot)$ based on the training data set Ω, such as k-fold crossvalidation accuracy [30], is used to guide the choice of K. Now the SVM parameter tuning problem can be formulated as the optimization problem

$$\begin{array}{ll} \max & \hat{a}(K) \\ \text{s.t.} & K \in \mathcal{K}. \end{array} \tag{4}$$

Many classes of kernels have been proposed in the literature. The simplest model for $\mathcal{K}$ is the one in which the kernel is assumed to be proportional to a fixed base kernel K_0, namely

$$\mathcal{K} = \{CK_0 : C > 0\}. \tag{5}$$

As K_0 one can take, for instance, the so-called *linear kernel*,

$$K^{\text{lin}}(x, z) = x^\top z,$$

yielding the standard SVM model, [9, 15, 27, 46, 47]. A very simple yet extremely powerful is the class of *Radial Basis Function* (RBF) kernels [15, 29],

$$\begin{array}{lcl} \mathcal{K} & = & \{CK_\sigma^{\text{RBF}} : C > 0, \sigma > 0\}, \\ K_\sigma^{\text{RBF}}(x, z) & = & \exp(-\sum_{i=1}^d (x_i - z_i)^2/\sigma), \end{array} \tag{6}$$

which has been extended by considering the scaling factor σ to be variable-dependent, yielding the *anisotropic* RBF model, see e.g. [12]. An alternative model studied, among others, in [1, 12, 22, 38, 37, 31, 39, 45], is the *Multiple Kernel Learning* (MKL) model. MKL is especially suitable when the data set has variables of different nature, calling for the use of different kernel models for the different types of variables involved. In its simplest version, R base kernels, $K_1, \ldots, K_R$, are given, and a conic combination is sought:

$$\mathcal{K} \;=\; \{\sum_{j=1}^R \mu_j K_j : \mu_j \geq 0 \quad \forall j = 1, 2, \ldots, R\}. \tag{7}$$

Such base kernels K_j may be, for instance, RBF kernels with different (but fixed) scaling factors σ_j for each j. While it is frequently claimed that the most relevant parameters to be tuned are the weights in the conic combination of kernels, [22], one may also consider to tune the kernels K_j, choosing them from different kernel sets $\mathcal{K}_j$, [22], yielding

$$\mathcal{K} \;=\; \{\textstyle\sum_{j=1}^{R} \mu_j K_j : \mu_j \geq 0, \quad K_j \in \mathcal{K}_j \quad \forall j = 1, \ldots, R\}. \tag{8}$$

This ends our review of the most popular kernel models in the literature. At this point, it is important to stress that, the richer the kernel class, the higher the value of the estimate $\hat{a}$, but this does not necessarily imply that the actual classification rate a also improves when the kernel class is enriched, due to the so-called overfitting phenomenon. This explains the variety of models, with different levels of generality, that can be found in the literature, and the need for a tuning method to be able to adapt to them.

To end with the structure of the tuning problem (4), we now discuss its objective function and the challenges when optimizing it. Some papers take as surrogate $\hat{a}(\cdot)$ of the accuracy $a(\cdot)$ a distribution-free, but kernel-specific, bound on the probability of misclassification, see [12, 18, 48]. While such functions $\hat{a}$ are usually smooth in the parameters, allowing for the use of high-order local search methods, other surrogates, not necessarily differentiable, have also been proposed, [3, 21, 50]. Most of the papers take as $\hat{a}$ the k-fold crossvalidation accuracy estimate, see [30]. This is also the approach taken in this paper. Note that in this case the cost of evaluating $\hat{a}$ is high. Indeed, evaluating $\hat{a}$ at a given set of parameter values amounts to solving k quadratic problems of the form (3). Also, local-search optimization methods might be not effective because the problem is multimodal, and these methods are challenged by the fact that the objective function is piecewise constant, and hence gradient-type information may be useless.

In this second part of the introduction, we review proposals to solve the resulting optimization problem. For simple kernel models, such as (5) with one single parameter C, the tuning is usually done by a grid search on a sufficiently big interval, say $[2^{-12}, 2^{12}]$. However, and due to the cost of evaluating the objective function, grid search is quite inefficient, becoming infeasible if the dimension of the parameter space is not too small, even if the grid is not too fine. Several heuristic algorithms have been proposed in the literature. Some are ad-hoc for a particular kernel model, such as [29], while others are metaheuristics.

An early reference on metaheuristics is [43], where a Pattern Search approach is introduced. An improvement is proposed in [2], where Simulated Annealing is used to screen the neighborhoods in [2].

Since [43], many other metaheuristics have been proposed in the literature. In [13], a genetic algorithm is used for parameter tuning within the RBF kernel model. Since the parameters are real-valued, a 0–1 encoding, of a given precision, is used. Alternative mutation and crossover operators for real-valued parameters are proposed in [36]. In [19], an evolutionary algorithm based on the so-called Covariance Matrix Adaptation Evolution Strategy, [24], is proposed.

In [20], the so-called Efficient Parameter Selection via Global Optimization algorithm is proposed. It is an iterative method based on estimating the objective function given its value in a collection of inspected solutions. This is done using an online gaussian process, whose parameters are chosen by maximum likelihood. As the authors point out, this method is only competitive when the dimension of the parameter space is low.

Other popular metaheuristic strategies such as Variable Neighborhood Search and Ant Colony Optimization have received perhaps less attention when tuning SVM parameters, [10, 51].

Most of existing approaches in the literature show their performance in the RBF kernel model (6), where only two parameters, C and σ, are to be tuned. An exception is [2], where the anisotropic RBF kernel model [12] is considered. This is a generalization of the RBF kernel model in which parameters C and σ_i $(i = 1, \ldots, d)$ need to be tuned. As is the case for the anisotropic RBF kernel, complex models, involving many parameters, can be seen as extensions of simpler and easy-to-tune models, yielding a nested sequence of models of increasing complexity.

In this paper we propose an algorithm which successfully exploits this *nested* property of complex methods, i.e., the ability to define a sequence of nested subproblems, with two main advantages. First, our framework is general enough to allow one to address, with the very same method, several popular SVM parameter models encountered in the literature. Indeed, to illustrate the versatility of our algorithm, we present experiments for an array of MKL models. MKL models have attracted a lot of attention and many ad-hoc approaches exist, see [22] for a through review on the most successful of these approaches. Second, as algorithmic requirements we only need a black box to train SVMs. In other words, as soon as an SVM package, such as LIBSVM [11], SVMTorch [14] or SVMlight [28], a general-purpose scientific computing, Statistics or Machine Learning package such as MATLAB, R, SAS or WEKA [49], or any routine for the minimization of convex quadratic functions under linear constraints is at hand, our approach is readily applicable. In contrast, some of the specialized MKL techniques we compare with require, for instance, Second-Order Cone Programming (SOCP) solvers, as in [1].

The remainder of the paper is structured as follows. In Section 2, we propose our nested heuristic, which is tested in Section 3 against benchmark methods for different kernel models. Concluding remarks and lines of future research are outlined in Section 4.

2 The nested heuristic

In this section we propose a nested heuristic for SVM parameter tuning, where we assume that a nested structure for the kernel model to be tuned and a metaheuristic are at hand. Below we discuss these two ingredients before presenting the algorithm.

2.1 Preliminaries

Complex models, involving many parameters, can be seen as extensions of simpler and easy-to-tune models, and therefore they can be considered as nested within another model. Throughout this section, we will assume that we have a series of nested kernel models, $\mathcal{K}_{(1)} \subset \mathcal{K}_{(2)} \subset \ldots \subset \mathcal{K}_{(H)} = \mathcal{K}$. For example, kernel model (8) can be embedded in a nested sequence $\mathcal{K} = \mathcal{K}_{(3)} \supset \mathcal{K}_{(2)} \supset \mathcal{K}_{(1)}$. Here, $\mathcal{K}_{(2)}$ corresponds to model (7), where all the individual kernels K_j have been fixed in advance, while $\mathcal{K}_{(1)}$ corresponds to model (5), with $K_0 = \sum_{j=1}^{R} K_j$, where in addition all weights μ_j are assumed to be equal to a common weight C. When solving the tuning problem (4) for $\mathcal{K}$, we will use a sequential approach where the (suboptimal) solution found for kernel model $\mathcal{K}_{(h)}$ becomes an initial solution to the tuning problem (4) for $\mathcal{K}_{(h+1)}$. Note that the optimization model associated with $\mathcal{K}_{(h)}$ is obtained by adding a set of constraints to (4).

In terms of metaheuristics, we have chosen the well-known Variable Neighborhood Search (VNS), but other metaheuristics can be used too. VNS [25, 41] sequentially moves through the feasible region by searching solutions in a neighborhood of the current best solution while, at the same time, systematically changes the size of the neighborhood to avoid getting trapped at local optima. It has been mainly applied to combinatorial optimization problems, see e.g. [25, 40], or continuous problems with a combinatorial structure [6, 5, 25], though it has also been recently proposed for continuous optimization problems, [8, 16, 17, 33, 42, 40]. In [10], a basic version of VNS for SVM parameter tuning can be found, together with the task of gene selection.

2.2 The algorithm

We now present a basic VNS, Algorithm 1, suitable for low dimension parameter space, see Figure 1. Algorithm 1 will be sequentially used in our nested heuristic, Algorithm 2, see Figure 2. In the following, and without loss of generality, we assume that the kernel model is parametrized by a p-dimensional parameter θ, i.e., $\mathcal{K}$ can be expressed as $\mathcal{K} = \{K(\theta) : \theta \in \Theta\}$, and the tuning problem can be formulated as maximizing $\hat{a}(K(\theta))$ over θ in Θ. For instance, in model (8), where the kernel classes $\mathcal{K}_j$ follow the RBF model (6), $\mathcal{K}$ is parametrized by $\theta = (\mu_1, \dots, \mu_R, \sigma_1, \dots, \sigma_R)$.
Apart from the maximum on the number iterations m, Algorithm 1 requires a norm $\|\cdot\|$ and thresholds $\{r_\kappa\}$ to define the neighborhood structure, and the maximum number of steps $\kappa_{\max}$ that should be performed without improvement on the objective function. In our computational section, we show that with straightforward choices, we obtain comparable accuracies to those reported by the well-known ad-hoc method by Keerthi and Lin [29] for the tuning of the RBF model.
When p is high, as it is usually the case when $\mathcal{K}$ is given by (8), there is little hope that this basic version of VNS, or any other metaheuristic, will reach the regions of good solutions, since the dimension of the search space is too large. In this case, we propose to use Algorithm 2, where the nested sequence $\mathcal{K} = \mathcal{K}_{(H)} \supset \dots \mathcal{K}_{(2)} \supset \mathcal{K}_{(1)}$ at hand is exploited. In our computational section, we show that a straightforward implementation of Algorithm 2 yields comparable accuracies to those reported by ad-hoc methods for the tuning of an array of MKL models.
The idea of embedding the optimization problem into a nest of simpler optimization problems is not new in the literature. For instance, [34, 35] propose a two-phase method for the molecular

```
Algorithm 1: Basic VNS algorithm for parameter tuning

INPUT: Kernel set $\mathcal{K} = \{K(\theta) : \theta \in \Theta\}$. Maximum number of iterations $m$. Neighborhood
       structure $\{N_1, N_2, \ldots, N_{\kappa_{\max}}\}$, with $N_\kappa(\tilde{\theta}) = \left\{\theta \in \Theta : \|\tilde{\theta} - \theta\| \leq r_\kappa\right\}$ and $0 < r_1 <$
       $r_2 < \ldots < r_{\kappa_{\max}}$.

Initialization: Select an initial solution $\tilde{\theta} \in \Theta$; set $\kappa \leftarrow 1$; set iter $\leftarrow 0$.

 Step 1. Repeat until iter $= m$ or $\kappa = \kappa_{\max}$ :

       Step 1.1. Shaking. Generate a random solution $\theta'$ in the $\kappa$-neighborhood of the
                 incumbent solution $\tilde{\theta}$, $\theta' \in N_\kappa(\tilde{\theta})$.

       Step 1.2. Neighborhood change. If $\hat{a}(\theta') > \hat{a}(\tilde{\theta})$, then move ($\tilde{\theta} \leftarrow \theta'$) and reset
                 the neighborhood ($\kappa \leftarrow 1$); otherwise, set $\kappa \leftarrow \kappa + 1$.

       Step 1.3. Set iter $\leftarrow$ iter $+ 1$.

 Step 2. If iter $= m$, STOP with solution $\tilde{\theta}$; otherwise, reset $\kappa \leftarrow 1$ and go to Step 1.
```

Figure 1: Pseudocode for Algorithm 1

shape optimization problem. The initial solution of the second phase is found using a modified objective function, a challenging task itself [23], which is based on prior knowledge of the properties of the global optima. The parameter tuning problem addressed in this paper lacks of this kind of prior knowledge, since the objective function is not known in a closed form. Hence, the approach in [34, 35] is not applicable here. Instead, our solution approach modifies (enlarges) the feasible set: we solve, by means of a heuristic, the tuning problem on the smaller kernel model $\mathcal{K}_{(h)}$, and use the (suboptimal) solution obtained as initial solution for $\mathcal{K}_{(h+1)}$, of higher dimension using the same heuristic.

3 Computational results

In this section we study the performance of our approach for the RBF kernel model (6) and the variants of the MKL kernel model (7) and (8) discussed in the introduction. For kernel

```
Algorithm 2: Nested VNS algorithm for parameter tuning

INPUT: Nested kernel models $\mathcal{K}_{(1)} \subset \mathcal{K}_{(2)} \subset \ldots \subset \mathcal{K}_{(H)} = \mathcal{K}$. Maximum number of iterations
       for each kernel model: $m_1, \ldots, m_H$.

Initialization: Set $h \leftarrow 1$. Randomly choose an initial solution $\tilde{\theta} \in \mathcal{K}_{(1)}$.

Step 1. Repeat while $h \leq H$.

    Step 1.1. Set the initial solution to $\theta \leftarrow \tilde{\theta}$.

    Step 1.2. Run Algorithm 1 for model $\mathcal{K}_{(h)}$ for a number of iterations $m_h$, yielding
              as output $\theta_h$.

    Step 1.3. Set $\tilde{\theta} \leftarrow \theta_h$ and $h = h + 1$.
```

Figure 2: Pseudocode for Algorithm 2

model (6), the performance of our algorithm is tested against the benchmark procedure by Keerthi and Lin [29]. For the MKL tests, we will use [22], which reviews the state-of-the-art in MKL and provides a very comprehensive computational experience including a wide variety of methods. Some of the ad-hoc methods for MKL reviewed and tested in [22] have high software requirements, including Second-Order Cone Programming (SOCP) solvers, [1]. In the following section we describe the data sets used in our experiments. The numerical results are presented in Sections 3.2 and 3.3.

The implementation of Algorithm 1 uses $\|\cdot\| = \|\cdot\|_\infty$, $r_\kappa = \kappa$ and $\kappa_{\max} = 25$. The choice of r_κ and $\kappa_{\max}$ is inspired in the grid approach which searches among the 25 values $-12, -11, \ldots, 12$. We use the publicly available package LIBSVM [11] to train SVM.

3.1 Data sets

In order to show results on the RBF kernel model, we use the 13 data sets in [44][1], which are widely used in the classification literature. Table 1 shows details on these data sets, including

[1]Available at `http://www.raetschlab.org/Members/raetsch/benchmark`

their name, the size of the training sample `tr`, the size of the testing sample `test`, and the number of predictor variables d. Rästch et al. [44] give 100 partitions of each data set into a training sample and a testing sample. In order to make a fair comparison, we use the same setup as in [29]. In particular, we only consider the first of those 100 partitions to compute the reported test error, and use 5–fold crossvalidation to compute the estimate $\hat{a}$.

name	`tr`	`test`	d
banana	400	4900	2
diabetis	468	300	8
image	1300	1010	18
splice	1000	2175	60
ringnorm	400	7000	20
twonorm	400	7000	20
waveform	400	4600	21
german	700	300	20
heart	170	100	13
thyroid	140	75	5
titanic	150	2051	3
flare-solar	666	400	9
breast-cancer	200	77	9

Table 1: Data sets for the RBF kernel tests

Table 2 shows details on the large data sets used in [22][2] for MKL kernel models, with similar information to the one found in Table 1. In these data sets, predictor variables are split into T clusters, $B_1, B_2, \ldots, B_T$; the last eight columns in Table 2 report on the sizes of the different clusters T, the total number of predictor variables d and the number of predictor variables in each cluster d_t, for $t = 1, 2, \ldots, T$. As already mentioned in the introduction, MKL is particularly appealing when such a clustering is known because models (7)-(8) can be used for a set of base kernels, where each kernel uses only predictor variables from one cluster.
Since we will benchmark our VNS against those methods surveyed in [22], we will use the same setup as in [22]. In particular, the test error was computed either using either (a) the testing set provided, or (b) one third of the data set, randomly chosen, the remaining two thirds being used as training set. A 5×2–fold is applied to compute the estimates $\hat{a}$.

[2]Available at `http://mkl.ucsd.edu/dataset/pendigits`, `http://archive.ics.uci.edu/ml/datasets/Multiple+Features` and `http://archive.ics.uci.edu/ml/datasets/Internet+Advertisements`

name	`tr`	`test`	T	d	d_t size of cluster t					
pendigitEO	7494	3498	4	352	16	16	64	256		
pendigitSL	7494	3498	4	352	16	16	64	256		
mfeatEO4	1333	667	4	427	76	64	240	47		
mfeatEO6	1333	667	6	649	76	64	240	47	216	6
mfeatSL4	1333	667	4	427	76	64	240	47		
mfeatSL6	1333	667	6	649	76	64	240	47	216	6
addata	2186	1093	5	1554	457	495	472	111	19	

Table 2: Data sets for the MKL tests

3.2 Results on the RBF model

In this section we present results on the tuning problem when the kernel class is the $\mathcal{K}^{\text{RBF}}$, as defined by (6). We have benchmarked our VNS algorithm against the procedure in Keerthi and Lin [29], called hereafter the KL method. In order to make a fair comparison with KL, and as in [29], a logarithmic scale is used, the search is restricted to the box $\Theta = [-8, 2] \times [-8, 8]$, the grid step size is 0.5, thus the total number of iterations is 54, i.e., $m = 54$ in Algorithm 1, and the initial solution is $(-3, 0)$. Note that an iteration has the same computational requirements in both methods. Since the dimension p of the parameter space Θ is low, namely, $p = 2$, we use the basic VNS given in Algorithm 1 for this experiment.

Table 3 shows results for KL and the basic VNS. For each method, we report its test error as a percentage, and the returned parameter θ. The errors are very similar, sometimes even equal. VNS obtains better results in 5 data sets, where the difference in errors ranges from 0.60% to 3.46%. KL is the best in 4 cases, where the difference in errors never exceeds 0.70%. In the remaining 4 data sets the error is exactly the same. In conclusion, the VNS error is never worse than that of KL by more than 0.70 percentage points, and therefore both methods yield comparable results.

3.3 Results on the Multiple Kernel Learning models

The data sets in [22] have a particular structure: the predictor variables are clustered into T groups $B_1, B_2, \ldots, B_T$; a kernel model is defined for each cluster, and the kernel is defined as a function of the T kernels involved. In this section, we consider MKL kernel models (7) and (8), in which the base kernels are either a linear or a RBF kernel applied to each cluster of

	KL			VNS		
name	error	$\log(C)$	$\log(\sigma)$	error	$\log(C)$	$\log(\sigma)$
banana	11.59	-2.00	-1.00	11.61	-2.07	-0.87
diabetis	24.00	4.00	6.00	24.67	-0.94	4.02
image	5.84	-0.50	3.00	2.38	1.26	0.80
splice	10.53	-0.50	6.00	9.93	1.90	6.62
ringnorm	1.44	-3.00	4.00	1.70	1.89	3.10
twonorm	2.47	-2.50	5.50	2.77	1.01	6.08
waveform	11.39	0.00	6.00	10.46	0.90	4.57
german	21.33	3.00	6.00	21.33	0.19	4.09
heart	21.00	-3.00	4.50	20.00	0.16	7.38
thyroid	5.33	0.00	-1.00	5.33	-0.55	-1.12
titanic	22.92	-2.00	3.50	22.92	-1.39	2.23
flare-solar	34.50	-0.50	3.50	34.50	0.04	3.32
breast-cancer	29.87	3.50	7.00	28.57	1.75	5.83

Table 3: Test errors and tuned parameters for the RBF kernel

variables B_t. We benchmark our VNS against the 12 linear combination methods reported in Gönen and Alpaydın [22].

The contribution of this section is two-fold. First, we show that the nested VNS is competitive for existing, in general more sophisticated and ad-hoc, methods in the literature for the same kernel models. To show this, we consider, as in [22], the kernel class defined in (7), taking as base kernels K_t the linear kernels (Section 3.3.1) and also the RBF kernels with fixed scaling factors σ_t for each t (Section 3.3.2). Second, we show that nested VNS can be also directly applied to more general models, such as (8), giving even better results in terms of accuracy. These results will be presented in Section 3.3.3. Note that the kernel class defined in model (8) cannot be addressed by the algorithms reviewed in [22], which are specific for (7). In contrast, nested VNS is a general approach that can handle many different models. The purpose of Section 3.3.3 is to illustrate the potential of this advantage in terms of classification accuracy.

3.3.1 Tuning the linear combination of fixed linear kernels

This section is devoted to tune a linear combination of T linear kernels of the form

$$K^{\text{lin}}_{B_t}(x,z) \;=\; \sum_{i\in B_t} x_i z_i, \tag{9}$$

i.e.,

$$\mathcal{K} \;=\; \{\textstyle\sum_{t=1}^{T} \mu_t K_{B_t}^{\mathrm{lin}} : \, \mu_t \geq 0 \quad \forall t = 1, 2, \ldots, T\}. \tag{10}$$

We use a nested sequence of models $\mathcal{K}_{(1)} \subset \mathcal{K}_{(2)} = \mathcal{K}$, where $\mathcal{K}_{(1)}$ is the one-dimensional model,

$$\mathcal{K}_{(1)} \;=\; \{\mu \textstyle\sum_{t=1}^{T} K_{B_t}^{\mathrm{lin}} : \, \mu \geq 0\}. \tag{11}$$

We run the nested VNS given in Algorithm 2, with a maximum number of iterations respectively of $m_1 = 100$ and $m_2 = 500$. The parametrization for $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ uses a logarithmic scale, as done in Section 3.2.

The results can be found in Table 4. The first column contains the name of the data set; the second column gives the dimension p of the parameter space Θ (in this case, $p = T$); the next three columns are devoted to the 12 benchmarking methods reviewed and tested in [22], which we will denote by 'InGonAlp', reporting the best, the median and the worst test error across them. The remaining columns show the results obtained with our nested VNS. The sixth column reports the test error of the nested VNS. The seventh column gives the ranking of the nested VNS among the 13 MKL methods at hand.

Although our nested VNS is not systematically the best, it beats the 12 benchmarking methods in two data sets and it behaves as second-best in another data set. In the remaining data sets VNS has always an accuracy within the range of the state-of-the-art methods and very close to the median.

		InGonAlp			Nested VNS	
name	p	best	median	worst	error	rank
pendigitEO	5	6.47	6.66	11.07	7.25	11/13
pendigitSL	5	8.88	9.06	15.56	10.45	11/13
mfeatEO4	5	1.99	2.15	4.22	2.34	10/13
mfeatEO6	7	1.61	1.76	3.10	1.46	1/13
mfeatSL4	5	4.82	5.11	9.46	6.19	12/13
mfeatSL6	7	2.19	2.54	10.82	2.30	2/13
addata	6	3.41	3.72	4.90	3.22	1/13

Table 4: Test errors for MKL with lineal kernels

3.3.2 Tuning the linear combination of fixed RBF kernels

In this section we analyze the results obtained when we tune a linear combination of $3T$ RBF base kernels of the form

$$K^{\mathrm{RBF}}_{B_t,\sigma}(x,z) \quad = \quad \exp(-\sum_{i\in B_t}(x_i - z_i)^2/\sigma), \tag{12}$$

where the scaling factor σ of each RBF kernel is fixed in advance. More precisely, the kernel model considered corresponds to the kernel class (7),

$$\mathcal{K} \quad = \quad \{\textstyle\sum_{t=1}^{T} \mu_{1t} K^{\mathrm{RBF}}_{B_t,\frac{d_t}{4}} + \mu_{2t} K^{\mathrm{RBF}}_{B_t,d_t} + \mu_{3t} K^{\mathrm{RBF}}_{B_t,4d_t} : \mu_{jt} \geq 0 \quad \forall t = 1,2,\ldots,T,\ j = 1,2,3\}, \tag{13}$$

where each base kernel $K^{\mathrm{RBF}}_{B_t,\sigma}$ has the form (12) for *fixed* scaling factor σ, namely, $\sigma \in \{\frac{d_t}{4}, d_t, 4d_t\}$, with d_t being the number of predictor variables in B_t.
The strategy to design a nested VNS is similar to the one used in Section 3.3.1. A nested sequence of models $\mathcal{K}_{(1)} \subset \mathcal{K}_{(2)} = \mathcal{K}$ is defined, where $\mathcal{K}_{(1)}$ is a one-dimensional model,

$$\mathcal{K}_{(1)} \quad = \quad \{\mu \textstyle\sum_{t=1}^{T} \left(K^{\mathrm{RBF}}_{B_t,\frac{d_t}{4}} + K^{\mathrm{RBF}}_{B_t,d_t} + K^{\mathrm{RBF}}_{B_t,4d_t} \right) : \mu \geq 0\}, \tag{14}$$

With this model structure, we run the nested VNS given in Algorithm 2, where the maximum number of iterations considered for each model are again $m_1 = 100$ and $m_2 = 500$.
As for the experiments in Section 3.2, the parameterization of $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ also use the logarithmic scale.
The results are presented in the first seven columns of Table 5[3]. The first column contains the name of the data set. Then there are six columns devoted to model (7). The first of these columns contains the dimension of the parameter space Θ (in this case, $p = 3T$). The next three columns report the best, the median and the worst test error across the 12 benchmarking methods. The next two columns report the test error and the ranking of our nested VNS.
Nested VNS accuracies are above the median on all five cases analyzed. In one data set, it even beats all benchmark methods, whereas in the others it performs very close to the best, being 0.15% the maximum difference with the best benchmark.

[3]Data sets pendigitEO and pendigitSL are not considered with RBF kernels, as is the case in [22]

	Model (7)						Model (8)		
	p	InGonAlp			Nested VNS		p	Nested VNS	
name		best	median	worst	error	rank		error	rank
mfeatEO4	12	0.67	0.96	2.18	0.74	4/13	8	0.72	4/13
mfeatEO6	18	0.58	0.67	7.22	0.65	3/13	12	0.53	1/13
mfeatSL4	12	1.43	1.63	4.60	1.58	5/13	8	1.40	1/13
mfeatSL6	18	0.97	1.25	7.25	0.99	3/13	12	0.95	1/13
addata	15	3.81	4.34	11.88	3.24	1/13	10	3.39	1/13

Table 5: Test errors for MKL with RBF kernels

Note that, throughout the rest of this section, the values of the scaling factors σ_t were chosen as in [22]. Our preliminary results using different values suggest that this choice is crucial to the accuracy of the resulting classifier. This suggests that a more general model, where the scaling factors σ_t are not fixed, but tuned by the algorithm (together with the rest of the parameters) may be a more suitable kernel model. The behavior of this model is studied in the next section.

3.3.3 Tuning all parameters

In this section we show how our VNS approach can seamlessly be adapted to solve the natural extension of model (7) given in (8): here the simultaneous tuning of T RBF kernels of type (12) and their scaling factors σ_t is considered. Note that none of the benchmark methods reviewed in [22] is directly applicable to this model, while it is straightforward to apply our nested VNS. Compared with the basic model (7), tuning the parameters of this more general model may lead to accuracy improvements, while the complexity of our procedure remains the same.

We use a nested sequence of models $\mathcal{K}_{(1)} \subset \mathcal{K}_{(2)}$. The outer class $\mathcal{K}_{(2)}$ is the kernel class (8), where each base kernel K_t is allowed to vary in the kernel class $\mathcal{K}_t = \left\{ K^{\text{RBF}}_{B_t,\sigma_t} : \sigma_t > 0 \right\}$. The inner class $\mathcal{K}_{(1)}$ considers that all the weights are equal, $\mu_t = \mu$, and all the scaling kernels are equal, $\sigma_t = \sigma$, i.e.:

$$\mathcal{K}_{(1)} = \left\{ \mu \sum_{t=1}^{T} K^{\text{RBF}}_{B_t,\sigma} : \mu > 0,\, \sigma > 0 \right\}.$$

The parametrization for $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ also uses the logarithmic scale, as done in Section 3.2. The nested VNS algorithm described in Algorithm 2 is run with $m_1 = 100$ and $m_2 = 500$.

The results are presented in the last three columns of Table 5. The first of these columns reports the dimension of the parameter space Θ, namely, $p = 2T$. The other two columns show the test

error and the rank of the nested VNS in model (8). We can observe that nested VNS applied to model (8) beats the best benchmark for model (7) in four out of five data sets, whereas in the other one it is better than the median and very close to the best.
This last experiment shows that, VNS, being a simpler method than many of the methods reviewed in [22], is able to successfully cope with more general kernel models than the methods existing in the literature.

4 Conclusions

In this work we have shown that a simple metaheuristic, namely, VNS, can be successfully customized to address the problem of parameter tuning in Support Vector Machines. The fact that parameter models are usually nested is successfully exploited in our version of VNS, called nested VNS, in which the parameters obtained as output when optimizing simpler models are used as starting solutions for tuning the parameters of the most general models. This key idea of exploiting the nested structure of models can be applied to many other metaheuristics, though we have found VNS easy to implement and tune. We conclude from our computational experience that, with simpler methods and using less demanding computational tools, we can get better or similar accuracy results than those obtained with benchmark procedures, which may be more specific and software demanding.
We emphasize that these encouraging results are obtained with a method, VNS, which is not specifically designed for MKL, as most of the benchmarking methods are. Moreover, VNS does not require specific software, hence it is more suitable for practitioners than the more specialized tuning techniques reviewed in [22], where some of the models call for specific algorithms, such as those needed to optimize a second-order cone programming problem [1].
In our VNS implementation no local searches are performed. This makes the method applicable under low software requirements (a numerical routine for solving convex quadratic optimization problems under linear constraints is sufficient) and applicable to parameter tuning problems for different kernel models. Nevertheless, embedding within our nested VNS as a local search methods such as those described in [12, 18, 48] deserves further study.
The strategy developed in this paper is also applicable to tune the parameters of SVM-like

models, such as those analyzed in [7], Support Vector Regression as well as other kernel models, [27]. Whether nested-VNS is competitive against benchmark procedures in these new settings deserves being explored.

Acknowledgments

The authors would like to thank the researchers M. Gönen and E. Alpaydın for kindly providing the results for the benchmarking methods used in Sections 3.3.2 and 3.3.3, which were not available in [22]. This work has been partially supported by projects MTM2012-36163-C06-03, ECO2011-25706 of Ministerio de Ciencia e Innovación and FQM-329 of Junta de Andalucía, Spain, both with EU European Development Funds.

References

[1] F.R. Bach, G.R.G. Lanckriet, and M.I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *ICML'04: Proceedings of the Twenty-First International Conference on Machine Learning*, page 6. ACM, New York, NY, 2004.

[2] A. Barbero Jiménez, J. López Lázaro, and J.R. Dorronsoro. Finding optimal model parameters by deterministic and annealed focused grid search. *Neurocomputing*, 72(13–15):2824–2832, 2009.

[3] M. Boardman and T. Trappenberg. A heuristic for free parameter optimization with support vector machines. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 610–617, 2006.

[4] B. Boser, I. Guyon, and V.N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, 1992.

[5] J. Brimberg, P. Hansen, N. Mladenović, and E.D. Taillard. Improvements and comparison of heuristics for solving the uncapacitated multisource weber problem. *Operations Research*, 48(3):444–460, 2000.

[6] J. Brimberg and N. Mladenović. A variable neighbourhood algorithm for solving the continuous location-allocation problem. *Studies in Locational Analysis*, 10:1–12, 1996.

[7] J.P. Brooks. Support vector machines with the ramp loss and the hard margin loss. *Operations Research*, 59(2):467–479, 2011.

[8] E. Carrizosa, M. Dražic, Z. Dražic, and N. Mladenović. Gaussian variable neighborhood search for continuous optimization. *Computers and Operations Research*, 39:2206–2213, 2012.

[9] E. Carrizosa and D. Romero Morales. Supervised classification and mathematical optimization. *Computers and Operations Research*, 40(1):150–165, 2013.

[10] K.Y. Chan, H.L. Zhu, M.E. Aydin, C.C. Lau, and H.Q. Wang. An integrated approach of support vector machine and variable neighborhood search for discovering combinational gene signatures in predicting chemo-response of osteosarcoma. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, pages 121–125, 2008.

[11] C.C. Chang and C.J. Lin. LIBSVM: A library for support vector machines. (Downloadable from website `http://www.csie.ntu.edu.tw/∼cjlin/libsvm`), 2001.

[12] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46:13–159, 2002.

[13] G. Cohen, M. Hilario, and A. Geissbuhler. Model selection for support vector classifiers via genetic algorithms. An application to medical decision support. In J.M. Barreiro, F. Martin-Sanchez, V. Maojo, and F. Sanz, editors, *Proceedings of ISBMDA 2004, Lecture Notes in Computer Science 3337*, pages 200–211. Springer, Berlin, 2004.

[14] R. Collobert and S. Bengio. SVMTorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160, 2001.

[15] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 2000.

[16] M. Drazić, V. Kovacevic-Vujcić, M. Cangalović, and N. Mladenović. GLOB: A new VNS–based software for global optimization. In L. Liberti, N. Maculan, and P. Pardalos, editors, *Global Optimization*, volume 84, pages 135–154. Springer US, 2006.

[17] M. Drazić, C. Lavor, N. Maculan, and N. Mladenović. A continuous variable neighborhood search heuristic for finding the three-dimensional structure of a molecule. *European Journal of Operational Research*, 185(3):1265–1273, 2008.

[18] K. Duan, S.S. Keerthi, and A.N. Poo. Evaluation of simple performance measures for tuning SVM hyperparameters. *Neurocomputing*, 51:41–59, 2003.

[19] F. Friedrichs and Ch. Igel. Evolutionary tuning of multiple SVM parameters. *Neurocomputing*, 64:107–117, 2005.

[20] H. Fröhlich and A. Zell. Efficient parameter selection for support vector machines in classification and regression via model-based global optimization. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1431–1438, 2005.

[21] C. Gold and P. Sollich. Model selection for support vector machine classification. *Neurocomputing*, 55(1-2):221–249, 2003.

[22] M. Gönen and E. Alpaydın. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, 12:2211–2268, 2011.

[23] A. Grosso, M. Locatelli, and F. Schoen. Solving molecular distance geometry problems by global optimization algorithms. *Computational Optimization and Applications*, 43(1):23–37, 2009.

[24] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[25] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[26] R. Herbrich. *Learning Kernel Classifiers. Theory and Algorithms.* MIT Press, 2002.

[27] T. Hofmann, B. Schölkopff, and A.J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3):1171–1220, 2008.

[28] T. Joachims. Making large–scale SVM learning practical. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184. The MIT Press, Cambridge, MA, 1999.

[29] S.S. Keerthi and Ch.-J. Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural Computation*, 15:1667–1689, 2003.

[30] R. Kohavi. Cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1137–1143. Morgan Kaufmann, 1995.

[31] G. Lanckriet, N. Cristianini, P. Bartlett, L. Ghaoui, and M.I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[32] N. Lavesson and P. Davidsson. Quantifying the impact of learning algorithm parameter tuning. In *AAAI'06: Proceedings of the 21st National Conference on Artificial Intelligence*, pages 395–400. AAAI Press, 2006.

[33] C. Lavor and N. Maculan. A function to test methods applied to global minimization of potential energy of molecules. *Numerical Algorithms*, 35:287–300, 2004.

[34] M. Locatelli and F. Schoen. Fast global optimization of difficult Lennard-Jones clusters. *Computational Optimization and Applications*, 21:55–70, 2002.

[35] M. Locatelli and F. Schoen. Efficient algorithms for large scale global optimization: Lennard-Jones clusters. *Computational Optimization and Applications*, 26:173–190, 2003.

[36] A.C. Lorena and A.C.P.L.F. de Carvalho. Evolutionary tuning of SVM parameter values in multiclass problems. *Neurocomputing*, 71:3326–3334, 2008.

[37] R. Luss. *Mathematical Programming for Statistical Learning with Applications in Biology and Finance.* PhD thesis, Princeton, 2009.

[38] R. Luss and A. d'Aspremont. Support vector machine classification with indefinite kernels. *Mathematical Programming Computation*, 1:97–118, 2009.

[39] I. Martín de Diego, A. Muñoz, and J.M. Moguerza. Methods for the combination of kernel matrices within a support vector framework. *Machine Learning*, 78(1–2):137–174, 2010.

[40] N. Mladenović, M. Dražic, V. Kovačevic-Vujčić, and M. Čangalović. General variable neighborhood search for the continuous optimization. *European Journal of Operational Research*, 191(3):753–770, 2008.

[41] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[42] N. Mladenović, J. Petrović, V. Kovačević-Vujčić, and M. Čangalović. Solving spread spectrum radar polyphase code design problem by tabu search and variable neighbourhood search. *European Journal of Operational Research*, 151(2):389–399, 2003.

[43] M. Momma and K. Bennett. A pattern search method for model selection of support vector regression. In *Proceedings of the SIAM International Conference on Data Mining*, pages 261–274, 2002.

[44] G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for adaboost. *Machine Learning*, 42:287–320, 2001.

[45] S. Sonnenburg, G. Rätsch, Ch. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7:1531–1565, 2006.

[46] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, 1995.

[47] V. Vapnik. *Statistical Learning Theory.* Wiley, 1998.

[48] V. Vapnik and O. Chapelle. Bounds on error expectation for support vector machines. *Neural Computation*, 12:2013–2036, 2000.

[49] Weka Machine Learning Project. University of Waikato, New Zealand. `http://www.cs.waikato.ac.nz/~ml/weka`.

[50] K.-P. Wu and S.-D. Wang. Choosing the kernel parameters for support vector machines by the inter-cluster distance in the feature space. *Pattern Recognition*, 42(5):710–717, 2009.

[51] X. Zhang, X. Chen, and Z. He. An ACO-based algorithm for parameter optimization of support vector machines. *Expert Systems with Applications*, 37(9):6618–6628, 2010.